PATENT ABSTRACTS OF JAPAN

(11)Publication number : 11-149380
(43)Date of publication of application : 02.06.1999

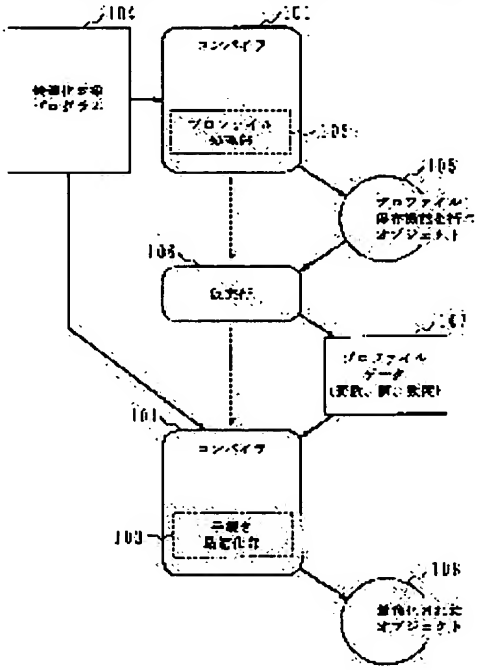
(51)Int.Cl. G06F 9/45

(21)Application number : 09-316738
(22)Date of filing : 18.11.1997

(71)Applicant : HITACHI LTD
REAL WORLD COMPUTING PARTNERSHIP
(72)Inventor : NISHIYAMA HIROYASU

(54) COMPILER, PROGRAM OPTIMIZING METHOD AND RECORDING MEDIUM RECORDING ITS PROCESSING PROGRAM

(57)Abstract:
PROBLEM TO BE SOLVED: To optimize a program even if procedure parameters are variables determined at the time of execution and to speed up execution of a program containing variables highly liable to take an identical value.
SOLUTION: A profile processing part 102 records a value which a variable in a program takes during execution and its frequency. A procedure optimization part 103 specializes a program part using the variable with the value (specified value) of a variable of high frequency and it is newly added. At the time of execution, it is discriminated whether the value of the variable is equal to the specified value or mot. When the values are equal, a compiler converts the program so that the specified program part which is newly added is read. The program containing the variable highly liable to take an identical value is executed on an existing processor at high speed.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)**公開特許公報**（Ａ）　(11)特許出願公開番号

特開平11－149380

(43)公開日　平成11年(1999)６月２日

(51)Int.Cl.[6]	識別記号	ＦＩ	
Ｇ０６Ｆ　9/45		Ｇ０６Ｆ　9/44	３２２Ｆ

審査請求　未請求　請求項の数７　ＯＬ　（全 11 頁）

(21)出願番号　特願平9－316738

(22)出願日　平成９年(1997)11月18日

(71)出願人　000005108
株式会社日立製作所
東京都千代田区神田駿河台四丁目６番地

(71)出願人　593162453
技術研究組合新情報処理開発機構
東京都千代田区東神田２－５－12　館角散ビル８階

(72)発明者　西山　博泰
神奈川県川崎市麻生区王禅寺1099番地　株式会社日立製作所システム開発研究所内

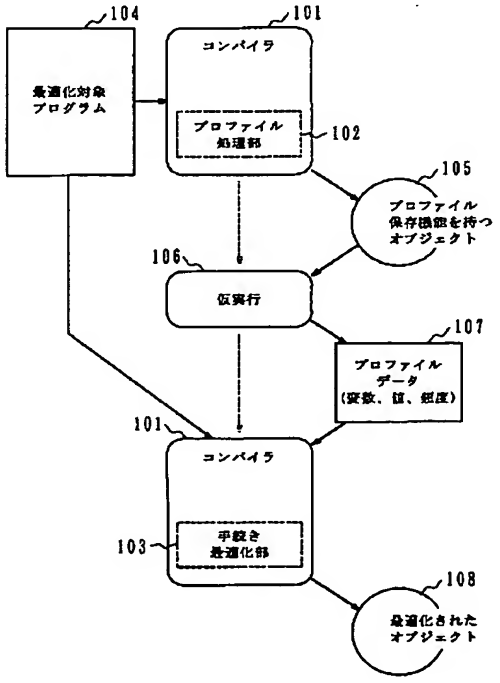
(74)代理人　弁理士　磯村　雅俊　（外１名）

(54)【発明の名称】　コンパイラとプログラム最適化方法およびその処理プログラムを記録した記録媒体

(57)【要約】

【課題】　手続きのパラメタが実行時に決定される変数である場合にもプログラムの最適化を可能とし、同一の値を取る可能性の高い変数を含んだプログラムを高速化させる。

【解決手段】　まず、プロファイル処理部１０２により、プログラム中の変数が実行中に取る値とその頻度とを記録させ、次に、手続き最適化部１０３により、変数を用いるプログラム部分を、高頻度の変数の値（特定値）で特殊化して新たに付加し、実行時、変数の値が特定値と等しいか否かを判別して、値が一致している場合に、新たに付加した特殊化プログラム部分を読み出すように、プログラムを変換するコンパイラ構成とし、同一の値をとる可能性の高い変数を含んだプログラムを、既存プロセッサ上で高速に実行させる。



【特許請求の範囲】
【請求項１】　プログラム言語で書かれたプログラムのコンパイル処理（翻訳）を行なうコンパイラであって、上記プログラム中の変数が取る値と該値を取る頻度とを対応付けて記録する手続きコード（プロファイルコード）を上記プログラムに付加する手段を有し、上記プログラムの実行時、上記変数が取った値と該値を取った頻度とを対応付けてプロファイル情報として記録するよう上記プログラムを翻訳することを特徴とするコンパイラ。
【請求項２】　請求項１に記載のコンパイラにおいて、上記プロファイルコードにより記録された所定の頻度以上の値（特定値）で上記変数を用いたプログラム部分を特殊化して上記プログラムに付加すると共に、上記変数が上記特定値を取った場合に該特定値で特殊化したプログラム部分を呼び出す手続きコード（分岐コード）を上記プログラムに付加する手段を有し、上記プログラムの実行時、上記変数が上記特定値を取った場合、上記特殊化したプログラム部分を呼び出すよう上記プログラムを翻訳することを特徴とするコンパイラ。
【請求項３】　請求項２に記載のコンパイラにおいて、上記プログラムの翻訳時、上記プロファイルコードを付加する手段を第１のオプションの指定に基づき起動する手段と、上記特殊化したプログラムを付加すると共に上記分岐コードを付加する手段を第２のオプションの指定に基づき起動する手段とを有し、上記第１のオプションが指定された状態で翻訳したプログラムを実行して上記プロファイル情報を記録した後、上記第２のオプションが指定された状態で上記プログラムを再度翻訳することを特徴とするコンパイラ。
【請求項４】　プログラム言語で書かれたプログラムのコンパイル処理（翻訳）を行なうコンパイラによる上記プログラムの最適化方法であって、上記プログラム中の変数が所定の頻度以上で同一の値（特定値）を取る場合、上記変数を用いたプログラム部分を上記特定値で特殊化したプログラムを選択するよう上記プログラムを翻訳することを特徴とするプログラム最適化方法。
【請求項５】　請求項４に記載のプログラム最適化方法において、上記プログラム中の変数が実行時に取る各値と該各値を取った頻度とを対応付けて記録する手続きコード（プロファイルコード）を上記プログラムに付加し、実行時の上記プロファイルコードによる記録情報（プロファイル情報）に基づき、上記特定値を特定することを特徴とするプログラム最適化方法。
【請求項６】　請求項５に記載のプログラム最適化方法において、上記特定値で上記変数を用いたプログラム部分を特殊化して上記プログラムに付加し、上記変数が上記特定値を取った場合に該特定値で特殊化したプログラム部分を呼び出す手続きコード（分岐コード）を上記プログラムに付加し、上記プログラムの実行時、上記変数の値が上記特定値と等しいか否かを判別し、等しい場合に上記特殊化したプログラム部分を呼び出すよう上記プログラムを翻訳することを特徴とするプログラム最適化方法。
【請求項７】　請求項４から請求項６のいずれかに記載のプログラム最適化方法の処理プログラムを記録したことを特徴とする記録媒体。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、ＣやＰＡＳＣＡＬ等のプログラム言語で書かれたプログラムのコンパイル処理（翻訳）を行なうコンパイラとそのプログラム最適化技術に係わり、特に、プログラムの実行速度を高速化するのに好適なコンパイラとプログラム最適化方法およびその処理プログラムを記録した記録媒体に関するものである。
【０００２】
【従来の技術】一般に、Ｃ言語における関数等、プログラム言語における手続きは、パラメタのうち幾つかが特定の値で呼び出されることが多い。これは、手続きが、汎用性を考慮して作成されることが多く、異なるパラメタで同一の手続き呼び出すことにより、同一のプログラムを利用して、異なる処理を実現できるようにしているためである。この結果、最初から特定の呼出しパターンのみに対応して記述した手続きと比較すると、その実行速度は低下してしまう。
【０００３】このような点に着目して、定数値をパラメタとして持つ手続き呼び出しを、パラメタの値を固定化したプログラムの呼び出しに変更することで、プログラムの実行速度を向上させるcloningと呼ばれる技術がある。ここで、元の手続きに対して、変更後のプログラムは、取り得るパラメタ値の範囲が限定されるので、このような手続きは、元の手続きに対して特殊化されているとされる。
【０００４】具体的には、例えば、K.D.Cooper,他による「“Procedure Cloning.” In Proceedings of the 1992 International Conference on Computer Languages, pages96-105, 1992」に示されているように、手続き呼び出しパラメタが定数の場合に、呼び出し点で定数値を持つパラメタに対応して手続き本体を複写し、パラメタの値を手続き間の定数伝搬によって定数に変更して特殊化した手続きを作成し、被呼び出し手続きを、この特殊化した手続きによって置き換える。このようなcloningを行なうには、手続き呼び出し部分のコンパイルを行なう際に、被呼び出し手続きの情報が必要となるので、コンパイル時に手続き間解析を行なって、cloningのための情報を保存しておく。
【０００５】同様の知見に基づいて、例えば、M.H.Lipasti,他による「“Exceeding the Dataflow Limit via Value Prediction.” In Proceedings of the 26th Annua

l International Symposium on Microarchitecture, pages226-237, 1996」では、値の局所性（value locality）という概念を導入し、ハードウェアによって命令レベルでの演算結果を予測することにより、処理を高速化するための技術が記載されている。値の局所性とは、同一の命令の計算結果が、ある一定の局所的な値をとる性質を表したものであり、上記技術では、この性質を利用して、命令アドレスと当該アドレスの演算やロードの結果の値の組を、「value prediction table」と呼ばれるプロセッサ上のテーブルに、ハードウェアによって記憶しておき、次に当該アドレスを実行した場合に、記憶しておいた値を利用することで、演算やロードに要するレイテンシを短縮する技術が記載されている。

【０００６】このレイテンシの短縮は、以前の演算結果による予測値を用いて、後続する演算を投機的に開始し、演算結果が予測値と等しくない場合には、投機的に実行した命令の実行をキャンセルすることにより実現されている。これにより、演算結果の予測が正しければ、演算を先行的に開始することができるので、プログラムを高速に実行することができる。尚、上記文献での評価結果によれば、ベンチマークプログラムに対して、平均６１％程度の値の局所性を持つことが示されている。

【０００７】このような値の局所性に基づいた手続き最適化のための技術とは別に、例えば、P.P.Chang,他による「“Using Profile Information to Assist Classic CodeOptimizations.” In SOFTWARE-PRACTICE AND EXPERIENCE, pages1301-1321, 1991」に記載されているように、プログラム内に存在する分岐の頻度を求め、これをプログラムの最適化に利用する技術がある。分岐の頻度を求めることにより、例えば、実行される確率の高いプログラム部分を求めることができるので、実行確率の高いパスを連続的に配置したり、実行頻度の高いパスの命令を優先的にスケジュールするといったコンパイラ最適化を適用することができる。

【０００８】このような最適化が効果を持つためには、高い分岐確率の予測精度が必要である。そこで、一度プログラムを実行し、そのときの分岐方向毎の分岐頻度をプロファイル情報として記憶しておき、次にコンパイルを行なう際に、この分岐頻度情報を参照して、実行頻度の高いパスを求める技術が利用されている。しかし、これらの従来技術においては、以下に示すような問題があった。

【０００９】まず、cloningを用いた技術では、特殊化の対象が、定数値をとる手続きのパラメタ、および、これを伝幡して定数となるプログラム断片に限定されていた。そのため、手続きのパラメタが実行時に決定される値である場合、手続きパラメタ値が常に同じであっても最適化が適用されなかった。また、cloningでは、手続きの実行頻度が考慮されないのでコードが増大し、手続き間解析を必要とするため、コンパイラの実現が複雑化する。

【００１０】尚、cloningのようなコンパイラによる最適化では、定数伝幡や定数畳み込みといった最適化を適用することにより、命令数の削減等、大きな効果を得ることができる。これに対して、M.H.Lipastiらによる技術では、レイテンシの削減効果しか得られない。また、その実現には複雑なハードウェア機構が必要となる。また、プロファイル情報を使用した上述のP.P.Changらによる技術においては、以前実行したプログラムで採取される情報は、分岐の実行頻度に限られていたため、コード移動やコード配置などに、その利用が限られていた。

【００１１】

【発明が解決しようとする課題】解決しようとする問題点は、従来の技術では、手続きのパラメタが実行時に決定される変数である場合にはプログラムを最適化できない点である。本発明の目的は、これら従来技術の課題を解決し、同一の値を取る可能性の高い変数を含んだプログラムを高速化させることを可能とするコンパイラとプログラム最適化方法およびその処理プログラムを記録した記録媒体を提供することである。

【００１２】

【課題を解決するための手段】上記目的を達成するため、本発明のコンパイラとプログラム最適化方法は、プログラム中の変数が実行中に取る値と、この値を取った頻度とを記録するための手続き（プロファイルコード）をコンパイル対象のプログラムに挿入して、プログラム実行時に変数が取った値とその頻度をプロファイル情報として記録し、この記録したプロファイル情報を、次回以降のコンパイル最適化処理で使用することにより、プログラムを高速化する。例えば、頻度が高く、プログラム中で変数が取り得る可能性の高い値（特定値）と、この特定値に変数の値を固定化することによって性能向上が期待できるプログラムの部分とを予測し、この予測したプログラム部分を特定値で特殊化したプログラム部分を新たに付加し、かつ、元のプログラムにおける、この特殊プログラム部分への制御の移行を行なう部分に、変数の値が予測した値（特定値）と等しいか否かを判別するコード（分岐コード）を挿入し、値が一致している場合に、変数の値を固定化した特殊プログラム部分を読み出すように、プログラムを変換する構成とする。このことにより、実行時に変数が特定値を取った場合のプログラムの実行速度を向上させることができ、同一の値（特定値）をとる可能性の高い変数を含んだプログラムを、既存プロセッサ上で高速に実行できる。

【００１３】

【発明の実施の形態】以下、本発明の実施例を、図面により詳細に説明する。図１は、本発明のコンパイラの構成とそのプログラム最適化に係わる動作の一実施例を示す説明図であり、図２は、図１におけるコンパイラを実行する計算機システムの構成例を示すブロック図であ

る。図２において、２０１はＣＰＵ（Central Processing Unit）からなるプロセッサ、２０２はＲＡＭ（Read Only Memory）からなる主記憶装置（図中、「主記憶」と記載）、２０３はＨＤＤ（Hard Disk Drive）からなる外部記憶装置（図中、「ディスク」と記載）、２０４は本発明のプログラム最適化を行なうコンパイラの処理プログラムを記録した光ディスク（図中、「ＯＤ」と記載）、２０５は光ディスク２０４からのデータの読み取りを行なう光ディスク駆動装置（図中、「ＯＤ駆動装置」と記載）である。

【００１４】光ディスク２０４に記録されたコンパイラは、光ディスク駆動装置２０５を介して外部記憶装置２０３に一旦格納され、さらに、主記憶装置２０２に読み出された後、プロセッサ２０１により実行される。コンパイラの翻訳（コンパイル処理）対象のプログラムも同様に、外部記憶装置２０３に格納され、主記憶装置２０２上に読み出され、コンパイラによりアプリケーション（オブジェクトプログラム）に変換（翻訳）され、外部記憶装置２０３に格納される。

【００１５】本例において、コンパイラは、図１に示すプロファイル処理部１０２と手続き最適化部１０３を有し、以下のようにして、本発明に係わるプログラム最適化を行なう。すなわち、まず、プログラム中に表れる変数（ｖａｒ）に対して、実行時にこの変数（ｖａｒ）が取った値（ｖａｌ）と、その値（ｖａｌ）を取った回数（ｃｏｕｎｔ）との組＜ｖａｌ，ｃｏｕｎｔ＞の集合を記憶するようプログラムを追加する。

【００１６】そして、次回、同じプログラムをコンパイルする際に、前回の実行の際に記憶した値（ｖａｌ）を参照し、同一の値（ｖａｌ）を取る頻度の高い変数（ｖａｒ）に対する手続き（ｆｕｎｃ）全体やその一部を、値（ｖａｌ）によって特殊化した手続き（ｆｕｎｃ’）を作成する。ここで、手続きを全て「ｆｕｎｃ」から「ｆｕｎｃ’」に、そのまま置き換えたのでは、変数（ｖａｒ）が「ｖａｌ」以外の値を取った場合に都合が悪いので、変数（ｖａｒ）が、「ｖａｌ」以外の値を取った場合に備えて、特殊化した手続き（ｆｕｎｃ’）を呼び出す前に、変数（ｖａｒ）の値が「ｖａｌ」と等しいか否かを検査し、等しい場合にのみ、特殊化した手続き（ｆｕｎｃ’）を呼び出すようにする。

【００１７】このようにしてプログラムを翻訳（コンパイル）することにより、従来のcloningでは、コンパイル時に値が定数であることをコンパイラで判定できた手続きのみが、値の特殊化による最適化の適用対象となっていたのに対して、値が常にあるいは多くの場合に同じであることが、コンパイル時に不確定なプログラムに対しても、値による特殊化を適用して、プログラムを高速化することが可能となる。すなわち、ここでの特定の値での手続きの特殊化を行なう際には、cloningのように、手続き呼び出し元の情報は必要なく、各変数の取り得る値とその頻度が分かれば良いので、手続き間解析が不可能な場合にも、最適化を適用することができる。

【００１８】このように、従来、分岐の頻度のみに対して適用されていた実行プロファイル情報の採取を、変数の取る値と、その頻度へ拡張することにより、コンパイラ最適化での利用範囲を拡大することができる。そして、このようなプロファイル情報の記憶、および、それを利用した最適化は、ソフトウェアのみで実現できるので、特別なハードウェアサポートを必要としない。

【００１９】以下、図１を用いて、このようなコンパイラの詳細説明を行なう。図１においては、実線矢印はデータの流れを表し、破線矢印はユーザによる操作の流れを表している。まず、プログラム作成者は、作成した最適化対象プログラム１０４を、コンパイラ１０１によってコンパイルする。この時、プログラム作成者は、プロファイル処理部１０２を起動するためのオプション（第１のオプション）を指定して、コンパイル指示操作を行なう。この結果、コンパイラ１０１は、プロファイル処理部１０２により、プログラム１０４中に現われる変数が実行時に取る値とその頻度をファイル等に記憶するプロファイル処理機能を、生成するオブジェクトに付加する。

【００２０】このようにして、プロファイル保存機能を持つオブジェクト１０５が生成される。このプロファイル保存機能を持つオブジェクト１０５（アプリケーション）を仮実行１０６することにより、プログラム１０４中に現れる変数に対して、その値と頻度を記録したプロファイル情報（図中、「プロファイルデータ」と記載）１０７が生成される。

【００２１】そして、次にプログラム１０４をコンパイルする際、プログラム作成者は、手続き最適化部１０３を起動するためのオプション（第２のオプション）を指定して、コンパイル指示操作を行なう。この結果、コンパイラ１０１は、手続き最適化部１０３により、先の実行により記憶されたプロファイル情報１０７を読み込み、特定の値を取る可能性の高い変数に対して、その値を取った場合に、当該値により特殊化された手続きあるいはプログラムの一部分が呼び出されるように最適化されたオブジェクト１０８を生成する。

【００２２】以下、このようなプログラムの最適化動作例を、図３～図８を用いて具体的に説明する。図３は、図１におけるプロファイル処理部で最適化対象プログラムに付加されたプロファイル情報の構造例を示す説明図である。プロファイル情報は、複数のプロファイルレコードから構成され、このプロファイルレコードは、各手続きのプロファイル対象変数毎に作成される。

【００２３】各プロファイルレコードに格納される情報は、図に示すように、プロファイル対象変数の所属する手続き名（ｐｒｏｃ）、変数名（ｎａｍｅ）、型（ｔｙｐｅ）、値（ｖａｌ）、変数が値ｖａｌを取った回数

（ｃｏｕｎｔ）などからなり、値（ｖａｌ）と回数（ｃｏｕｎｔ）の組については、頻度の高いものを複数個保存しておく。尚、この例では、値（ｖａｌ）の型を整数型（ｉｎｔ）としているが、符号小数点型などに関しても拡張して良い。また、値（ｖａｌ）と回数（ｃｏｕｎｔ）の組は、最も頻度の高い組１つのみを格納するようにすることでも良い。

【００２４】次に、図４と図５により、本発明の特徴的な処理を行なう図１におけるプロファイル処理部１０２と手続き最適化部１０３の動作について説明する。図４は、図１におけるプロファイル処理部の処理動作例を示すフローチャートである。プロファイル処理部１０２は、処理４０１より処理を開始し、処理４０２において、プロファイル対象変数の集合Ｖを求める。

【００２５】ここで、集合Ｖに求める変数は、プログラム中に現れる全ての変数としても良いが、全ての変数を対象とした場合は、プロファイル情報の量が膨大となる可能性があるため、別の値を取る可能性の高いことがコンパイル時に予測できる変数や、プロファイルを取っても、得られる最適化効果が薄い変数などについては、プロファイル対象変数から除いても良い。

【００２６】次に、処理４０３では、処理４０２において求めたプロファイル対象変数の集合Ｖが空集合か否かを判別する。空集合であれば、処理４０７に移り、全ての処理を終了するが、空集合でなければ、処理４０４において、集合Ｖより変数を１つ取り出して変数ｖとして格納する。続いて、変数ｖの定義点の集合Ｄを求める。尚、プログラム中の定義点の集合は、例えば、Ａｈｏ他による「Compilers-Principles,Techniques, and Tools,Addision-Wesley, 1996」に記載されているような、既知のプログラム解析技術を適用することにより求めることができる。

【００２７】次に、処理４０５において、求めた定義点の集合Ｄが空集合か否かを確かめ、空き集合であれば処理４０３に制御を移して次の変数を処理する。また、空き集合でなければ処理４０６に制御を移し、まず、集合Ｄより変数ｖの定義点を１つ取り出し、ｄ（変数）へ格納する。次に、ｄの定義点の直後に、その時点での変数ｖの値と、その値を取った回数を記録するコードを挿入し、続いて、処理４０５へ制御を移して、次の定義点を処理する。

【００２８】図５は、図１における手続き最適化部の処理動作例を示すフローチャートである。手続き最適化部１０３は、処理５０１により処理を開始し、処理５０２において、プロファイル情報の集合Ｐを求める。ここで、プロファイル情報の集合は、図４のプロファイル処理を行なう手続きコードを付加してコンパイルしたアプリケーションを仮実行等した際に記録されたものであり、これを、コンパイラ中に読み込むことで求められる。

【００２９】続いて、処理５０３では、処理５０２において求めたプロファイル情報の集合Ｐが空き集合か否かを確かめる。空き集合であれば、処理５０７へ制御を移して処理を終了する。また、集合Ｐが空集合でなければ、処理５０４へ制御を移し、まず、集合Ｐよりプロファイル対象変数を１つ取り出し、変数ｖへ格納する。次に、変数ｖが最も高い頻度で取る値を変数ｎへ、変数ｖが値ｎを取る確率を変数ｆへ格納する。

【００３０】次に、処理５０５において、処理５０４で求めた確率ｆが十分大きいか否かを確かめ、十分でない場合には、処理５０３へ制御を移して次の変数を処理し、十分大きければ処理５０６の制御に移る。処理５０６では、変数ｖが使用されるプログラム上の領域を変数Ｒへ求める。ここで、ある変数が使用されているプログラム上の領域についても、前記文献に述べられているような、既知のプログラム解析技術を適用することにより求めることができる。

【００３１】次に、領域Ｒを複写し、複写したコード上での変数ｖの使用を値ｎで置き換えた領域Ｒ’を作成する。さらに、領域Ｒへの制御フローに対して、変数ｖの値がｎか否かを確かめて、ｎと等しい場合にはＲ’へ、等しくない場合にはＲへ制御を移すようプログラムを変更する。続いて、処理５０３へ制御を移して、別の変数を処理する。

【００３２】このような図４、図５の処理による本発明の具体的なプログラム最適化例を、図６～図８を用い説明する。図６は、図１における最適化対象プログラムの具体例を示す説明図である。このプログラムにおいて、関数６０１（ｓｕｍ）のパラメタｘは、関数６０２（ｒｅａｄ＿ｉｎｔ）の呼び出し結果となっているため、従来のcloningでは、「ｒｅａｄ＿ｉｎｔ」が手続き間解析可能で、かつ、その返値が特定の定数であることがコンパイラによって保証できなければ、最適化を適用することができない。

【００３３】そこでまず、このプログラムを、変数の値とその値を取った頻度をプロファイル情報として記憶する機能を持ったプログラムへ、図１のプロファイル処理部１０１の図４における処理により変換する。すなわち、図４の処理４０２における処理で、プロファイル対象変数の集合を求める。ここで、図６のプログラムでは、関数ｓｕｍの変数ｉおよびｓについては、関数を構成するループ中でその値が不変ではないので、プロファイル情報を取る必要がない。また、関数ｍａｉｎの変数ｙについては、値の定義後にコンパイル時にソースコードの得られないシステムライブラリの呼び出しパラメタとして使用されるだけであるので、この値を特殊化しても最適化効果が得られる見込がない。そこで、このプログラム例では、変数ｘのみを対象とし、プロファイル対象集合Ｖ＝｛ｘ｝とする。

【００３４】次に、図４の処理４０３における処理で、

Ｖが空集合であるか否か確かめる。ここでは、Ｖは空集合でないので、図４の処理４０４へ制御を移す。図４の処理４０４では、集合Ｖより変数「ｘ」を取り出し、これを変数ｖへ格納する。また、Ｄに変数「ｘ」の定義点の集合を求める。ここでは、変数ｘの定義点は関数６０２の文である。次に、図４の処理４０５の処理において、Ｄが空集合か否かを確かめる。Ｄは空集合ではないので、図４の処理４０６へ処理を移す。

【００３５】図４の処理４０６では、Ｄより変数ｄに、変数ｘの定義点である関数６０２の文を取り出し、この関数６０２の文の直後に、変数ｘの取った値と、その値を取った回数を記録するプロファイルコードを挿入する。次に、図４の処理４０４でＤが空集合になったので、処理４０３へ制御を移すが、このとき、同様にＶも空集合になったので、処理４０７へ制御を移して、プロファイル処理を終了する。この結果得られるコードを図７に示す。

【００３６】図７は、図１のプログラム処理部により図６のプログラムに本発明に係わるプロファイルコードが付加されたプログラム例を示す説明図である。本例では、図４のプロファイル処理により、プロファイル情報を記憶するための処理関数（＿ｐｒｏｆｉｌｅ）を呼出すプロファイルコード７０１が、プログラム中に付加されている。ここで、このプロファイル情報の記録機能を付加したプログラムを実行してプロファイル情報を生成し、その結果、関数（ｒｅａｄ＿ｉｎｔ）の返値である変数ｘの値が「１０」である頻度が高いことが判明したとする。

【００３７】この場合の、この情報を利用しての、図１の最適化処理部１０３による図６のプログラムの最適化動作を、図５の処理に基づき具体的に説明する。まず、図５の処理５０２の処理では、Ｐにプロファイル情報の集合を求めるが、ここでは、プロファイルを記録した変数はｘのみであるので、ｘの情報のみがＰに求められる。次に、図５の処理５０３でＰが空集合か否かを確かめる。ここでは、Ｐは空集合ではないので、図５の処理５０４へ制御を移す。処理５０４では、Ｐより変数ｘを取り出し、Ｖへ格納する。

【００３８】同様に、ｖが最も高い頻度で取る値である「１０」を変数ｎへ格納し、値「１０」を取った確率をｆへ求める。次に、図５の処理５０５で、求めた確率ｆが十分大きいか否かを確かめる。この例では、ｆの値が十分大きいものとし、図５の処理５０６へ制御を移す。図５の処理５０６においては、Ｒに変数ｘの使用があるプログラム上の領域を求める。図６の例では、関数６０１（ｓｕｍ）のパラメタとして変数ｘが使用されているので、Ｒは、関数６０１（ｓｕｍ）の呼出し文とｓｕｍの定義となる。

【００３９】次に、Ｒ’にＲを複写し、変数ｘの使用を値「１０」で置き換える。ここで、関数ｓｕｍを複写し、パラメタを置き換えた関数（ｓｕｍ＿１０）のパラメタの値が「１０」で置き換えられるので、ループ解消や定数伝幡、静的評価といった最適化を適用することにより、「ｓｕｍ＿１０」を最適化することができる。次に、領域Ｒの制御の入口に、ｘの値が「１０」と等しい場合にＲ’へ、また、等しくない場合にＲへ制御を移動する分岐コードを付加する。図６の例では、Ｒの制御の入口は、関数６０１の文の直前なので、関数６０１の文の直前に、値の判別検査を行なうコードを挿入する。

【００４０】そして、図５の処理５０３では、Ｐが空集合となったので、処理５０７へ制御を移して処理を終了する。以上のような最適化を適用した結果、図６のプログラムは、図８に示すようなプログラムとなる。図８は、図１における手続き最適化部により図６のプログラムに本発明に係わる分岐コードと特殊化されたプログラムが付加されたプログラム例を示す説明図である。

【００４１】図８のプログラムでは、図６における関数６０１の文が、「ｉｆ（ｘ　＝＝１０）｛ｙ＝ｓｕｍ＿１０（）；｝ｅｌｓｅ｛ｙ＝ｓｕｍ（ｘ）；｝」となり、関数ｍａｉｎ中でｒｅａｄ＿ｉｎｔの結果が「１０」と等しいか否かを確認し、等しい場合に、関数ｓｕｍのパラメタの値を「１０」で特殊化した関数ｓｕｍ＿１０を呼出し、また、等しくない場合には、元の関数ｓｕｍを呼び出すようにプログラムが変更されている。そして、特殊化された関数ｓｕｍ＿１０（）には、ｒｅｔｕｒｎ（５５）から直ちに返値「５５」が渡される。

【００４２】このように、特殊化された関数ｓｕｍ＿１０（）は、パラメタｎを「１０」に特殊化することにより、ループが解消され、さらに、定数伝幡および定数畳み込み等の最適化により演算も除去されるので、元の関数ｓｕｍを実行するよりも効率が良い。このため、ｒｅａｄ＿ｉｎｔの結果が「１０」であることが多ければ、プログラムの実行は高速化されることになる。

【００４３】以上、図１～図８を用いて説明したように、本実施例のコンパイラでは、コンパイル対象のプログラム中の変数が、所定の頻度以上で同一の値（特定値）を取る場合、その変数を用いたプログラム部分を特定値で特殊化したプログラムを選択するようプログラムを翻訳し、プログラムの最適化を行なう。すなわち、第１のオプションを指定してコンパイルしたプログラムを仮実行させ、プログラム中の変数の値と、その値を取った頻度をプロファイル情報として記憶させる。そして、そのプロファイル情報を、次回以降の最適化処理で使用することにより、プログラムを高速化することができる。

【００４４】例えば、頻度が高く、プログラム中で変数が取り得る可能性の高い値を特定し、この特定した値で、変数を用いるプログラム部分を特殊化することにより、実行時に変数が特定値を取った場合の実行速度を向上させる。このことにより、同一の値をとる可能性の高

い変数を含んだプログラムを、既存プロセッサ上で高速に実行することができる。

【００４５】尚、本発明は、図１～図８を用いて説明した実施例に限定されるものではなく、その要旨を逸脱しない範囲において種々変更可能である。例えば、本実施例では、プログラム中に現われる変数のみをプロファイル情報として記憶する対象としたが、例えば、コンパイラ生成変数、プログラム中の分岐の条件、ループ中で参照される配列のオーバラップ関係などの変数に拡張することも可能である。また、コンパイラを記録するものとして、本例での光ディスクの他に、フレキシブルディスク（ＦＤ）等を用いても良い。

【００４６】

【発明の効果】本発明によれば、手続きのパラメタが実行時に決定される変数である場合にもプログラムを最適化して変換でき、同一の値を取る可能性の高い変数を含んだプログラムを高速化させることが可能である。

【図面の簡単な説明】

【図１】本発明のコンパイラの構成とそのプログラム最適化に係わる動作の一実施例を示す説明図である。

【図２】図１におけるコンパイラを実行する計算機システムの構成例を示すブロック図である。

【図３】図１におけるプロファイル処理部で最適化対象プログラムに付加されたプロファイル情報の構造例を示す説明図である。

【図４】図１におけるプロファイル処理部の処理動作例を示すフローチャートである。

【図５】図１における手続き最適化部の処理動作例を示すフローチャートである。

【図６】図１における最適化対象プログラムの具体例を示す説明図である。

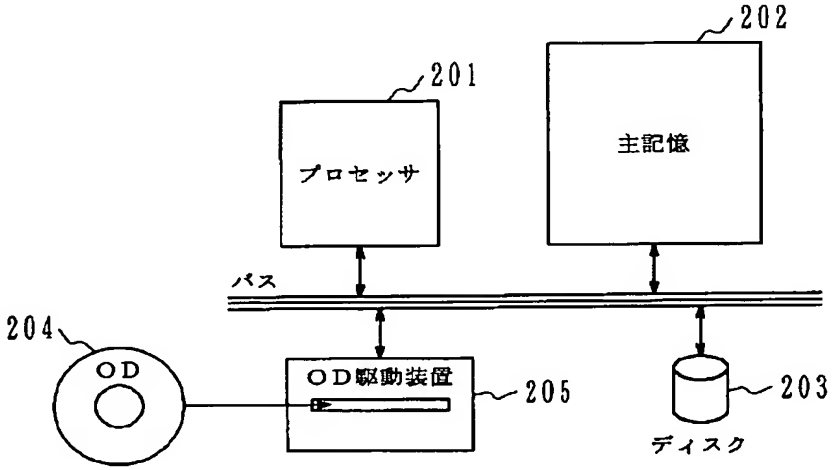
【図７】図１のプログラム処理部により図６のプログラムに本発明に係わるプロファイルコードが付加されたプログラム例を示す説明図である。

【図８】図１における手続き最適化部により図６のプログラムに本発明に係わる分岐コードと特殊化されたプログラムが付加されたプログラム例を示す説明図である。

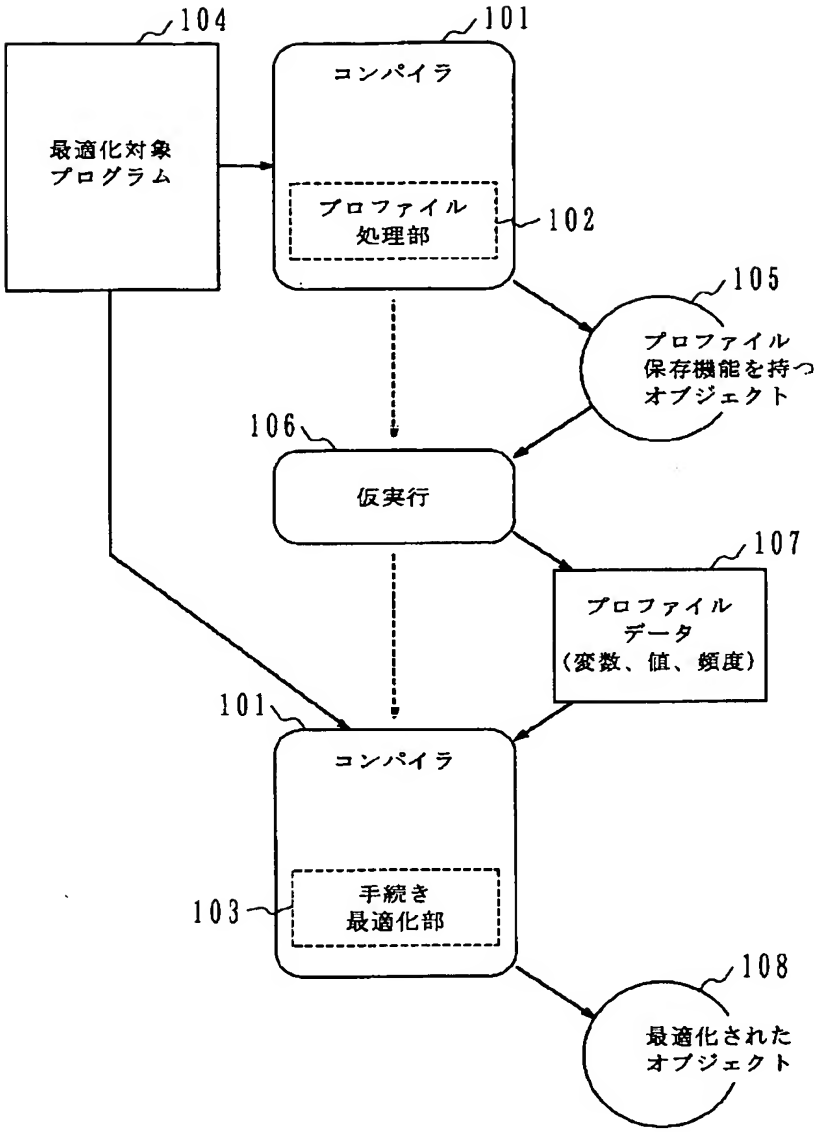
【符号の説明】

１０１：コンパイラ、１０２：プロファイル処理部、１０３：手続き最適化部、１０４：最適化対象プログラム、１０５：プロファイル保存機能を持つオブジェクト、１０６：仮実行、１０７：プロファイル情報、１０８：最適化されたオブジェクト、２０１：プロセッサ、２０２：主記憶装置、２０３：外部記憶装置、２０４：光ディスク、２０５：光ディスク駆動装置、６０１，６０２：関数、７０１：プロファイルコード。

【図２】



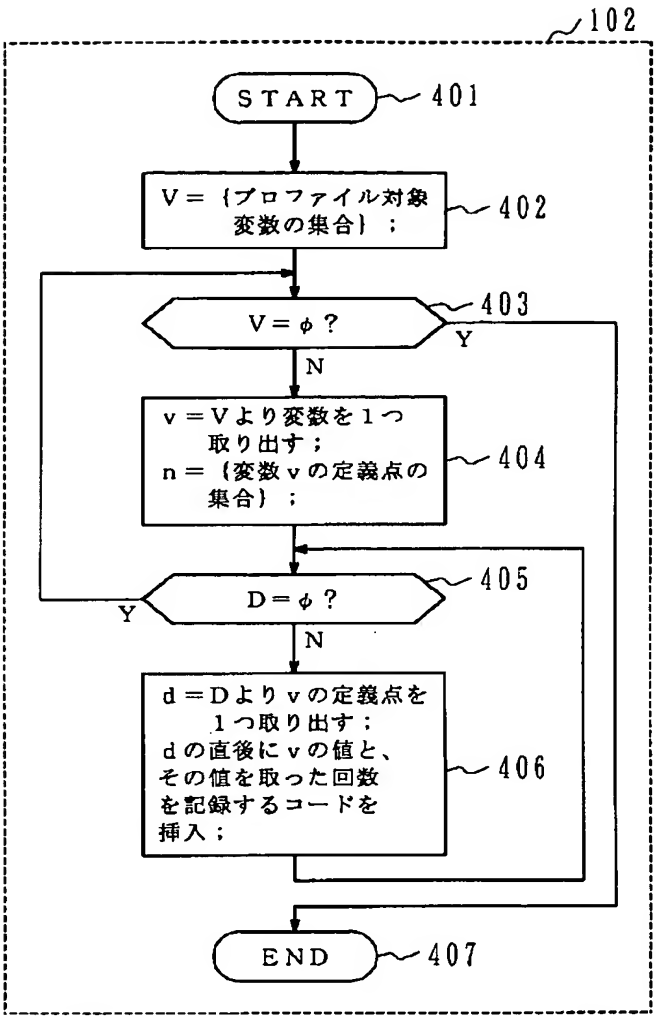
【図１】



【図３】

```
typedef struct {
   char proc[MAX_NAME];
   char name[MAX_NAME];
   int type;
   struct {
      int val;
      int count;
   } values[MAX_VALUE];
} ProfileRecord;
```

【図4】



【図6】

```
int  sum(int  n)
{
   int  i,   s  =  0;

   for(i  =  1;  i  <=  n;  i++)
      s  +=  i;
   return(s);
}

main( )
{
   int  x, y;

   x  =  read_int( );
   y  =  sum(x);
   printf("%d\n",   y);
}
```

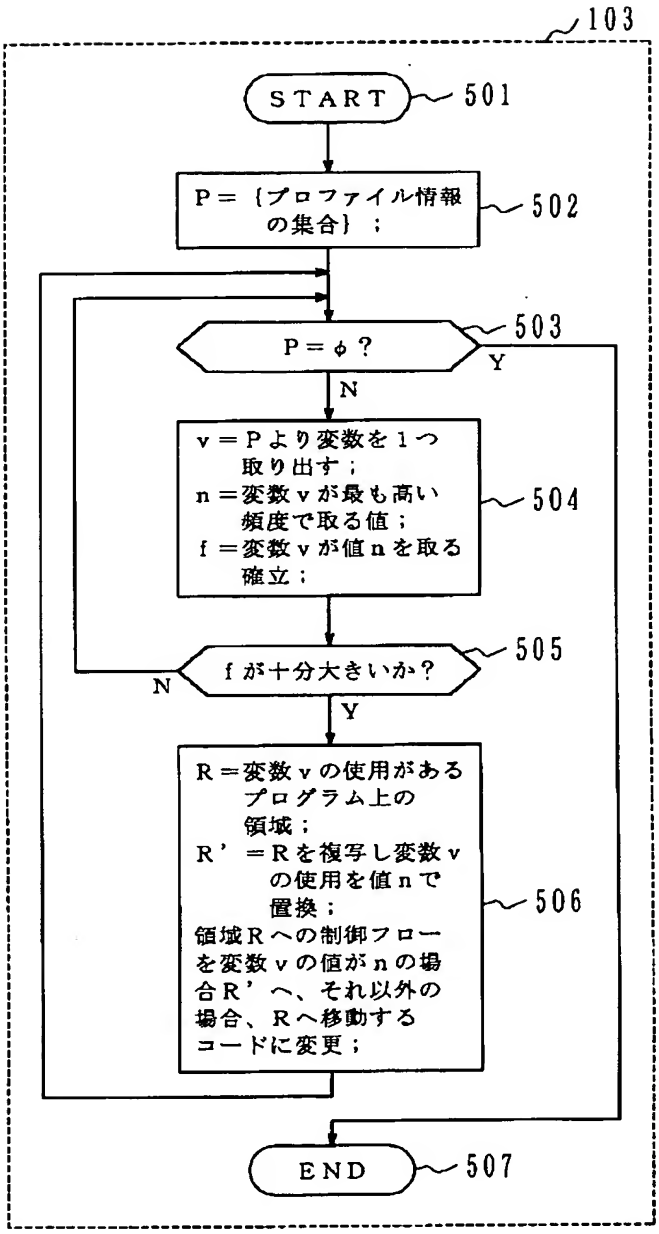
602

601

【図8】

```
int  sum(int  n)
{
   int  i,   s  =  0;

   for(i  =  1;  i  <=  n;  i++)
      s  +=  i;
   return(s);
}

int  sum_10( )
{
   return(55);
}

main( )
{
   int  x, y;

   x  =  read_int( );
   if(x  ==  10)  {
      y  =  sum_10( );
   }
   else  {
      y  =  sum(x);
   }
   printf("%d\n",   y);
}
```

【図5】


103
START
501
P＝｛プロファイル情報の集合｝；
502
P＝φ？
503
Y
N
v＝Pより変数を1つ取り出す；
n＝変数vが最も高い頻度で取る値；
f＝変数vが値nを取る確立；
504
f が十分大きいか？
505
N
Y
R＝変数vの使用があるプログラム上の領域；
R’＝Rを複写し変数vの使用を値nで置換；
領域Rへの制御フローを変数vの値がnの場合R’へ、それ以外の場合、Rへ移動するコードに変更；
506
END
507

【図7】

```
int sum(int n)
{
   int i, s = 0;

   for (i = 1; i <= n; i++)
      s += i;
   return (s);
}

main( )
{
   int x, y;

   x = read_int( );
   _profile("main", "x", x);
   y = sum(x);
   printf("%d\n", y);
}
```

701